

Introduction to R

Part I/II



Version 10

Juni 2026

Alexander Hirner & Aleksandra Tyjan

R - Part I

Version 10 (June 2026)

Alexander Hirner

Table of contents

1	Getting Started	2
2	The Most Important R Objects (Overview)	4
3	Data Managment (<code>data.frame</code> Objects in Detail)	8
4	Selected Functions	16
5	Groupwise Descriptive Statistics	21
6	Graphs	23
7	Summarizing Data Using <code>summarytools</code>	29
8	Functions and Control Flow by Example	32
9	Recommended Reading	35

1 Getting Started

1.1 Installation

1. Download R from <https://www.r-project.org>.
2. Install the tidyverse package by typing in `install.packages("tidyverse")`.
3. Install an editor. Recommendations:
 - **VSCodium** from <https://vscodium.com/> or
 - **RStudio** from <https://www.rstudio.com/>.

Order of Installation

If you are using RStudio, the order of installation (first R, then the editor) matters (otherwise you will get an error during installation).

1.2 Recommended Packages

- tidyverse - **A collection of packages for data science.**
- ggplot2 - Elegant data visualisations using *The Grammar of Graphics*.
- XLConnect - Excel connector for R.
- foreign - Read data stored by Minitab, SPSS, SAS, etc.
- dplyr - A grammar of data manipulation.
- tidyr - Tidy messy data.
- data.table - Extension of data.frame (fast!)
- doBy - Groupwise Statistics, LSmeans, Linear Estimates, Utilities.
- cowplot - Various features to create publication-quality figures (e.g. combining plots).
- questionr - Functions to Make Surveys Processing Easier.
- summarytools - Tools to Quickly and Neatly Summarize Data.

Installing Packages

The command to install additional packages is `install.packages("...")`. However, it is possible that the editor which you are using, supports you, i.e. in RStudio you can click on **Tools** and **Install packages** (and then simply follow the instructions).

Functions from previously installed packages can be invoked by

1. using the syntax `packagename::functionname(...)` or
2. “loading” the package using `library(packagename)` or `require(packagename)` (preferably at the beginning of a script) which makes the prefix `packagename::` unnecessary then (to “remove” the package again use `detach`).

Note that each way has its advantages and disadvantages.

1.3 Basic R Commands

Let us start with the “basics” first, e.g. creating simple objects, inspecting and deleting them, managing file paths and getting help:

Command	Description
<code>getwd()</code>	Gets working directory.
<code>setwd("path")</code>	Sets working directory (R uses forward slashes in file paths!).
<code>variable <- value</code>	Assigning a value to a variable.
<code>variable <- c(val1, val2, ...)</code>	Assigning values to a variable (c is for <i>combine</i>).
<code>ls()</code>	Lists all objects in the workspace.
<code>rm(object)</code>	Removes an object.
<code>help()</code>	Getting help.
<code>?object</code>	An alias for <code>help()</code> .
<code>help.search("searchstring")</code>	Searches the help system for a given string.
<code>??searchstring</code>	An alias for <code>help.search()</code> .
<code>#</code>	The hash sign introduces a comment.
<code>data()</code>	Lists built-in sample data.

Functions can also be nested (i.e. “a function in a function”), e.g. the command `rm(list=ls())` deletes everything from memory.

Using `rm(list=ls())`

The command `rm(list=ls())` is helpful to clean up the memory before starting a new script or project. However, use it with caution since there is no “undo” for this command.

2 The Most Important R Objects (Overview)

2.1 Vectors

Usage and Important Operations

Vectors can be used as “containers” to store univariate data.

Command	Description
<code>x <- c(val1, val2, ...)</code>	Creates a vector. Use quotes for string literals.
<code>x[i]</code>	Value on position i of vector x . The first element is on position 1 in R.
<code>x[-i]</code>	All values except for the one on position i .
<code>x[j:k]</code>	Observations from position j to position k .
<code>x[c(j,m)]</code>	Observations on position j and m .
<code>length(x)</code>	Number of Observations in x .
<code>rev(x)</code>	Reverses elements of x .
<code>x <- seq(from, to, by)</code>	Generates an arithmetic progression. Use <code>1:n</code> as an alias for <code>seq(1,n,1)</code> .
<code>x <- rep(pattern, n)</code>	Repeats a pattern n times

About R Objects

Commands like `x[-1]` or `rev(x)` never change the object *in situ* - these commands (when used in interactive mode) simply print out the values to the console. If you want to change the vector permanently, the assignment operator (`<-`) must be used.

Example

```
# a vector:
height <- c(180, 167, 198.5, 156, 170, 172, 169, 155)
height <- height[-3] # drop the third entry
summary(height) # summary statistics
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
155.0	161.5	169.0	167.0	171.0	180.0

2.2 Objects of the data.frame-Class

Usage and Important Operations

A `data.frame` is the appropriate structure to store multivariate data, i.e. data consisting of **n** rows (observations) and **k** columns (variables or data fields).

Command	Description
<code>X <- data.frame(v1, v2, ...)</code>	Creates a <code>data.frame</code> . Here, v1 and v2 are vectors of the same length .
<code>X[i,]</code>	Row (observation) i .
<code>X[, k]</code>	Column (variable) k .
<code>X[i:j,]</code>	Observations from row i to j (and all variables).
<code>X[c(i,k),]</code>	Observations in row i and k (and all variables).
<code>X[-c(i,k),]</code>	All observations except those in row i and k (and all variables).
<code>X[, c(i,k)]</code>	All rows, only column i and k .
<code>X[, -c(i,k)]</code>	All rows, drop values in column i and k .
<code>X\$varname</code>	Addresses a column by its name.
<code>names(X)</code>	Gets (or assigns) variable names.

Example

In the following example we will create a small `data.frame` with `n=5` observations (cats) and `k=3` variables from scratch:

```
id <- c(1,2,3,4,5) # or 1:5
sex <- c("male", "female", "female", "female", "male")
weight <- c(3, 4, 3.7, 2.5, 0.9)
cats <- data.frame(id, sex, weight)
rm(id, sex, weight) # drop original vectors now!
```

This is our data:

```
print(cats)

  id  sex weight
1  1 male   3.0
2  2 female  4.0
3  3 female  3.7
4  4 female  2.5
5  5 male   0.9
```

Now let us address one of the columns, i.e. “weight” (weight of the cat in kilos) to calculate their average:

```
mean(cats$weight)

[1] 2.82
```

Now let us drop the column “id” because we do not need it and display the `data.frame`:

```
cats <- cats[ , -1]
cats
```

```
      sex weight
1  male    3.0
2 female    4.0
3 female    3.7
4 female    2.5
5  male    0.9
```

2.3 Lists

Usage and Important Operations

- Lists in R are capable of storing different data types (strings, numbers, vectors, `data.frames` or even lists).
- If you want to create a list “from scratch”, use the function `list` and simply pass the objects that you want to store in the list. With `names()` it is possible to assign names to the elements stored in the list.
- Whenever you run a statistical procedure (e.g. ANOVA, linear regression, cluster analysis, ...) the output **should always be stored in a variable**. This variable is often a list (or at least “behaves” like a list). List elements can be addressed by using the `$`-symbol or double square brackets and an index.
- Note that a `data.frame` is a special kind of a list - it is a list where all elements are vectors of the same length.

Example

```
v1 <- c(1, 2, 3)
v2 <- c("a", "b", "c")
v3 <- data.frame(v1, v2)
v4 <- 7
v5 <- "hello world"
mylist <- list(v1, v2, v3, v4, v5)
names(mylist) <- c("v1", "v2", "DF", "a_number", "a_string")
```

We can now address single elements from the list:

```
print(mylist$v1)
```

```
[1] 1 2 3
```

```
print(mylist$a_string)
```

```
[1] "hello world"
```

2.4 Matrices

Usage

You might need the `matrix` data type if you do linear algebra with R.

Example

```
x <- seq(1,12,1) # or simply x <- 1:12
y <- matrix(x, nrow=3, byrow=TRUE)
print(y)
```

```
      [,1] [,2] [,3] [,4]
[1,]     1     2     3     4
[2,]     5     6     7     8
[3,]     9    10    11    12
```

In the example above, a matrix with 12 elements (numbers from 1 to 12) and three rows (`nrow`) was created. Alternatively, you can specify `ncol` (the number of columns). The flag `byrow` indicates whether the matrix should be filled up “row-wise” (if `TRUE`) or “column-wise” (if `FALSE`).

With `t(X)` you can transpose a matrix `X`. This command also works for objects of the `data.frame`-class.

2.5 Type Conversions

Sometimes the following type conversion functions can be helpful:

- `as.numeric`
- `as.character`
- `as.vector`
- `as.matrix`
- `as.data.frame`

2.6 More Data Types

There are more data types (i.e. there are objects of the `tibble`-class and some more). These data types require additional packages but their behaviour and purpose (storing multivariate data) is similar to a `data.frame`.

3 Data Managment (`data.frame` Objects in Detail)

3.1 Logical Operators

For some operations (e.g. filtering data or creating new variables) we need comparison operators:

Command	Description
<code><</code>	less than
<code><=</code>	less than or equal to
<code>></code>	greater than
<code>>=</code>	greater than or equal to
<code>==</code>	equality
<code>!=</code>	inequality
<code>%in%</code>	match with one element of a vector?

These operators can be combined using `&` (logical AND) or `|` (logical OR).

3.2 Working with Files

Command	Description
<code>read.table</code>	Reads a file and creates a <code>data.frame</code> from it.
<code>read.csv</code>	Reads a file and creates a <code>data.frame</code> from it.
<code>read.csv2</code>	Reads a file and creates a <code>data.frame</code> from it.
<code>write.table</code>	Prints a <code>data.frame</code> to a file.
<code>write.csv</code>	Prints a <code>data.frame</code> to a file.
<code>write.csv2</code>	Prints a <code>data.frame</code> to a file.

If you want to import data from an Excel-file directly (i.e. without saving it as a csv-file), you can use the function `readWorksheetFromFile` from the library `XLConnect`.

Possible pitfalls when dealing with files

- Make sure you are in the right directory. You might need the `setwd`-command.
- The functions listed above differ with respect to their default settings (i.e. for the field separator string and the decimal symbol). Use the `help` function to find out details.
- Commands such as `write.table` (or related functions) overwrite existing files **without warning (!)**.

3.3 Data Managment Operations in Detail

Command	Description
<code>dim(X)</code>	Displays number of rows and columns.
<code>head(X, n)</code>	Returns the first n rows of X .
<code>tail(X, n)</code>	Returns the last n rows of X .
<code>str(X)</code>	Compactly displays structure of X .
<code>Z <- rbind(X, Y)</code>	Adding observations from X and Y .
<code>Z <- cbind(X, Y)</code>	Adding a column to X .
<code>X\$new <- formula</code>	Creates a new column in X with a formula .
<code>X <- transform(X, new1=f1, new2=f2,...)</code>	Calculates new columns new1 and new2 with the formula f1 or f2 , respectively.
<code>names(X)</code>	Displays (or changes) variable names.
<code>ifelse(cond, ifpart, elsepart)</code>	Conditional execution (i.e. for recoding).
<code>Y <- subset(X, conditions)</code>	Creates a subset Y based on logical conditions.
<code>X <- X[order(X\$var),]</code>	Sorts your data.frame by one or more variables. Use decreasing=TRUE for descending order.
<code>Y <- split(X, X\$splitvar)</code>	Creates a list of data.frame-objects (for each value of splitvar).
<code>Y <- merge(x, y, by.x, by.y)</code>	Merges (i.e. joins) data.frame-objects by a common column.

3.4 Example

Loading Packages

```
library(tidyverse) # for dplyr and tidyr-functions
```

Generating Random Data

```
set.seed(123)
N <- 20
id <- sprintf("P%03d", 1:N)
sex <- sample(c("m", "w"), N, replace = TRUE)
age <- sample(18:90, N, replace = TRUE)
weight <- round(rnorm(N, mean = 75, sd = 12), 1)
height <- round(rnorm(N, mean = 172, sd = 10), 0)
haem <- round(rnorm(N, mean = 14, sd = 1.5), 1)
creat <- round(rnorm(N, mean = 1.0, sd = 0.2), 2)
leuk <- round(rnorm(N, mean = 6.5, sd = 1.5), 1)
bg <- sample(c("A", "B", "AB", "zero"), N, replace=TRUE)
mydata <- data.frame(id, sex, age, weight, height,
                     haem, creat, leuk, bg)
```

Display the data.frame

```
print(mydata)
```

	id	sex	age	weight	height	haem	creat	leuk	bg
1	P001	m	74	76.8	167	14.1	1.09	4.0	A
2	P002	m	26	61.3	180	15.4	0.93	5.9	A
3	P003	m	89	90.0	171	17.1	1.23	7.9	AB
4	P004	w	43	80.1	175	13.3	1.20	5.6	AB
5	P005	m	24	71.5	172	10.5	1.11	7.4	B
6	P006	w	59	85.7	172	15.5	1.05	4.1	A
7	P007	w	26	85.5	186	12.9	0.87	6.4	zero
8	P008	w	53	84.9	170	13.0	1.27	7.3	B
9	P009	m	60	83.3	187	15.5	0.88	7.0	zero
10	P010	m	32	81.6	157	13.6	1.44	6.7	AB
11	P011	w	49	74.3	178	12.2	1.31	5.5	B
12	P012	w	24	71.3	173	14.3	0.95	5.2	AB
13	P013	w	26	70.4	174	13.8	0.79	5.0	B
14	P014	m	58	66.7	176	14.0	0.86	6.7	AB
15	P015	w	40	72.5	167	14.6	1.05	5.1	A
16	P016	m	44	59.8	169	13.4	0.95	5.8	A
17	P017	w	77	101.0	162	15.0	0.93	6.1	A
18	P018	m	70	89.5	161	13.7	0.81	9.3	A
19	P019	m	24	61.5	175	14.5	0.99	5.5	AB
20	P020	m	70	70.2	176	15.6	0.84	6.9	B

Information about a data.frame

```
head(mydata, 4)
```

	id	sex	age	weight	height	haem	creat	leuk	bg
1	P001	m	74	76.8	167	14.1	1.09	4.0	A
2	P002	m	26	61.3	180	15.4	0.93	5.9	A
3	P003	m	89	90.0	171	17.1	1.23	7.9	AB
4	P004	w	43	80.1	175	13.3	1.20	5.6	AB

```
tail(mydata, 4)
```

	id	sex	age	weight	height	haem	creat	leuk	bg
17	P017	w	77	101.0	162	15.0	0.93	6.1	A
18	P018	m	70	89.5	161	13.7	0.81	9.3	A
19	P019	m	24	61.5	175	14.5	0.99	5.5	AB
20	P020	m	70	70.2	176	15.6	0.84	6.9	B

```
str(mydata)
```

```
'data.frame': 20 obs. of 9 variables:
 $ id      : chr  "P001" "P002" "P003" "P004" ...
 $ sex     : chr  "m" "m" "m" "w" ...
 $ age     : int   74 26 89 43 24 59 26 53 60 32 ...
 $ weight  : num   76.8 61.3 90 80.1 71.5 85.7 85.5 84.9 83.3 81.6 ...
 $ height  : num   167 180 171 175 172 172 186 170 187 157 ...
 $ haem    : num   14.1 15.4 17.1 13.3 10.5 15.5 12.9 13 15.5 13.6 ...
 $ creat   : num    1.09 0.93 1.23 1.2 1.11 1.05 0.87 1.27 0.88 1.44 ...
 $ leuk    : num     4 5.9 7.9 5.6 7.4 4.1 6.4 7.3 7 6.7 ...
 $ bg      : chr   "A" "A" "AB" "AB" ...
```

```
glimpse(mydata) # from tidyverse
```

```
Rows: 20
```

```
Columns: 9
```

```
$ id      <chr> "P001", "P002", "P003", "P004", "P005", "P006", "P007", "P008", ~
$ sex     <chr> "m", "m", "m", "w", "m", "w", "w", "w", "m", "m", "w", "w", "w" ~
$ age     <int> 74, 26, 89, 43, 24, 59, 26, 53, 60, 32, 49, 24, 26, 58, 40, 44, ~
$ weight  <dbl> 76.8, 61.3, 90.0, 80.1, 71.5, 85.7, 85.5, 84.9, 83.3, 81.6, 74.~
$ height  <dbl> 167, 180, 171, 175, 172, 172, 186, 170, 187, 157, 178, 173, 174~
$ haem    <dbl> 14.1, 15.4, 17.1, 13.3, 10.5, 15.5, 12.9, 13.0, 15.5, 13.6, 12.~
$ creat   <dbl> 1.09, 0.93, 1.23, 1.20, 1.11, 1.05, 0.87, 1.27, 0.88, 1.44, 1.3~
$ leuk    <dbl> 4.0, 5.9, 7.9, 5.6, 7.4, 4.1, 6.4, 7.3, 7.0, 6.7, 5.5, 5.2, 5.0~
$ bg      <chr> "A", "A", "AB", "AB", "B", "A", "zero", "B", "zero", "AB", "B", ~
```

```
dim(mydata)
```

```
[1] 20 9
```

Creating a New Column

```
X <- mydata
```

```
head(X, 2)
```

```
  id sex age weight height haem creat leuk bg
1 P001  m  74   76.8   167 14.1  1.09  4.0  A
2 P002  m  26   61.3   180 15.4  0.93  5.9  A
```

```
X$bmi <- X$weight/(X$height/100)^2
```

```
head(X, 3)
```

```
  id sex age weight height haem creat leuk bg      bmi
1 P001  m  74   76.8   167 14.1  1.09  4.0  A 27.53774
2 P002  m  26   61.3   180 15.4  0.93  5.9  A 18.91975
3 P003  m  89   90.0   171 17.1  1.23  7.9 AB 30.77870
```

Creating New Columns Using transform

```
X <- mydata
head(X, 3)
```

```
   id sex age weight height haem creat leuk bg
1 P001  m  74   76.8    167 14.1  1.09  4.0  A
2 P002  m  26   61.3    180 15.4  0.93  5.9  A
3 P003  m  89   90.0    171 17.1  1.23  7.9 AB
```

```
X <- transform(X, bmi = weight/(height/100)^2, heightm = height/100)
head(X, 3)
```

```
   id sex age weight height haem creat leuk bg      bmi heightm
1 P001  m  74   76.8    167 14.1  1.09  4.0  A 27.53774    1.67
2 P002  m  26   61.3    180 15.4  0.93  5.9  A 18.91975    1.80
3 P003  m  89   90.0    171 17.1  1.23  7.9 AB 30.77870    1.71
```

Using subset

```
X <- mydata
X1 <- subset(X, sex == "w" & weight < 72)
X1
```

```
   id sex age weight height haem creat leuk bg
12 P012  w  24   71.3    173 14.3  0.95  5.2 AB
13 P013  w  26   70.4    174 13.8  0.79  5.0  B
```

```
X <- mydata
X2 <- subset(X, id %in% c("P001", "P006", "P012"))
X2
```

```
   id sex age weight height haem creat leuk bg
1  P001  m  74   76.8    167 14.1  1.09  4.0  A
6  P006  w  59   85.7    172 15.5  1.05  4.1  A
12 P012  w  24   71.3    173 14.3  0.95  5.2 AB
```

Using split

```
X <- mydata
by_bg <- split(X, X$bg)
# loop through all data frames in the list by_bg
# and return the first 2 rows of each data.frame
lapply(by_bg, head, 2)
```

\$A

	id	sex	age	weight	height	haem	creat	leuk	bg
1	P001	m	74	76.8	167	14.1	1.09	4.0	A
2	P002	m	26	61.3	180	15.4	0.93	5.9	A

\$AB

	id	sex	age	weight	height	haem	creat	leuk	bg
3	P003	m	89	90.0	171	17.1	1.23	7.9	AB
4	P004	w	43	80.1	175	13.3	1.20	5.6	AB

\$B

	id	sex	age	weight	height	haem	creat	leuk	bg
5	P005	m	24	71.5	172	10.5	1.11	7.4	B
8	P008	w	53	84.9	170	13.0	1.27	7.3	B

\$zero

	id	sex	age	weight	height	haem	creat	leuk	bg
7	P007	w	26	85.5	186	12.9	0.87	6.4	zero
9	P009	m	60	83.3	187	15.5	0.88	7.0	zero

Binning a Variable into Categories

```
X <- mydata
summary(X$weight)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
59.80	70.35	75.55	76.89	85.05	101.00

```
X$weightcat <- cut(X$weight, breaks=c(-Inf, 65, 75, Inf), labels=c("cat1", "cat2", "cat3"))
head(X)
```

	id	sex	age	weight	height	haem	creat	leuk	bg	weightcat
1	P001	m	74	76.8	167	14.1	1.09	4.0	A	cat3
2	P002	m	26	61.3	180	15.4	0.93	5.9	A	cat1
3	P003	m	89	90.0	171	17.1	1.23	7.9	AB	cat3
4	P004	w	43	80.1	175	13.3	1.20	5.6	AB	cat3
5	P005	m	24	71.5	172	10.5	1.11	7.4	B	cat2
6	P006	w	59	85.7	172	15.5	1.05	4.1	A	cat3

Categorical Data (Recoding)

```
X <- mydata
X$bgroup <- recode(X$bg, "A" = "Group A", "B" = "Group B", "AB" = "Group AB", "zero" = "0")
head(X)
```

	id	sex	age	weight	height	haem	creat	leuk	bg	bgroup
1	P001	m	74	76.8	167	14.1	1.09	4.0	A	Group A
2	P002	m	26	61.3	180	15.4	0.93	5.9	A	Group A
3	P003	m	89	90.0	171	17.1	1.23	7.9	AB	Group AB
4	P004	w	43	80.1	175	13.3	1.20	5.6	AB	Group AB
5	P005	m	24	71.5	172	10.5	1.11	7.4	B	Group B
6	P006	w	59	85.7	172	15.5	1.05	4.1	A	Group A

Sorting Data

Sorting - Example 1

Sort the data by *bg* in ascending, then by *weight* in descending order:

```
X <- mydata
X <- X[order(X$bg, -X$weight), ]
head(X, 3)
```

	id	sex	age	weight	height	haem	creat	leuk	bg
17	P017	w	77	101.0	162	15.0	0.93	6.1	A
18	P018	m	70	89.5	161	13.7	0.81	9.3	A
6	P006	w	59	85.7	172	15.5	1.05	4.1	A

Sorting - Example 2

User-defined sorting

```
X <- mydata
myorder <- c("zero", "A", "B", "AB")

X <- X[order(match(X$bg, myorder)), ]
head(X, 5)
```

	id	sex	age	weight	height	haem	creat	leuk	bg
7	P007	w	26	85.5	186	12.9	0.87	6.4	zero
9	P009	m	60	83.3	187	15.5	0.88	7.0	zero
1	P001	m	74	76.8	167	14.1	1.09	4.0	A
2	P002	m	26	61.3	180	15.4	0.93	5.9	A
6	P006	w	59	85.7	172	15.5	1.05	4.1	A

Selecting Variables

Example: Keep columns 1, 2 and 7 only:

```
X <- mydata[, c(1, 2, 7)]
head(X, 2)
```

```
      id sex creat
1 P001   m  1.09
2 P002   m  0.93
```

Using rbind

rbind adds rows (observations):

```
X <- mydata
Xmale <- subset(X, sex == "m")
Xfemale <- subset(X, sex == "w")
Xtotal <- rbind(Xmale, Xfemale)
```

```
head(Xtotal, 2)
```

```
      id sex age weight height haem creat leuk bg
1 P001   m  74   76.8    167 14.1   1.09  4.0  A
2 P002   m  26   61.3    180 15.4   0.93  5.9  A
```

```
tail(Xtotal, 2)
```

```
      id sex age weight height haem creat leuk bg
15 P015   w  40   72.5    167 14.6   1.05  5.1  A
17 P017   w  77  101.0    162 15.0   0.93  6.1  A
```

Using cbind

cbind adds existing columns to a data.frame

```
X <- mydata
head(X, 2)
```

```
      id sex age weight height haem creat leuk bg
1 P001   m  74   76.8    167 14.1   1.09  4.0  A
2 P002   m  26   61.3    180 15.4   0.93  5.9  A
```

```
origin <- rep("randomly generated data", 20)
```

```
X <- cbind(X, origin)
```

```
head(X, 2)
```

```
      id sex age weight height haem creat leuk bg              origin
1 P001   m  74   76.8    167 14.1   1.09  4.0  A randomly generated data
2 P002   m  26   61.3    180 15.4   0.93  5.9  A randomly generated data
```

Reading and Writing Files

You can read/write files using the functions `read.table`, `read.csv`, `read.csv2` or `write.table`, `write.csv`, `write.csv2`, respectively.

4 Selected Functions

4.1 Preliminary Remarks

Applying Functions to Columns

Applying functions to columns

Functions can also be applied to whole vectors of data (columns in your `data.frame`) - in contrast to other programming languages, there is no need to use loops here.

Possible Pitfalls

Possible pitfalls

- The function `log()` is the “natural” logarithm.
- Trigonometric functions (and inverse trigonometric functions) return/expect the angle in radians (not in degrees!)

4.2 Common Functions in R

Function	Description
<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code>	Basic math.
<code>a^b</code> or <code>a**b</code>	Returns a^b .
<code>sqrt(x)</code>	$\sqrt{x}$
<code>pi</code> , <code>exp(1)</code>	Mathematical constants.
<code>sin(x)</code> , <code>cos(x)</code> , <code>tan(x)</code>	Trigonometric Functions.
<code>asin(x)</code> , <code>acos(x)</code> , <code>atan(x)</code>	Inverse trigonometric functions.
<code>floor(x)</code> , <code>ceiling(x)</code>	Rounding to the next integer.
<code>round(x, digits)</code>	Rounding to a given precision.
<code>log(x)</code>	(natural) logarithm.
<code>log10(x)</code>	logarithm (base 10).
<code>exp(x)</code>	Returns e^x .
<code>sum(x)</code>	Returns the sum of all elements in vector x .
<code>cumsum(x)</code>	Returns the cumulated sum of all elements in vector x .
<code>prod(x)</code>	Returns the product of all elements in vector x .
<code>cumprod(x)</code>	Returns the cumulated product of all elements in vector x .
<code>paste(x, y)</code>	Concatenates strings.
<code>class(x)</code>	Object classes.
<code>cat(x, y)</code>	Concatenates and prints.
<code>unique(x)</code>	Removes duplicates.
<code>set.seed(seed)</code>	Sets seed for random number generation.

4.3 Using any, all and which

Functions

Function	Description
<code>any(...)</code>	Are some values TRUE?
<code>all(...)</code>	Are all values TRUE?
<code>which(...)</code>	Which indices are TRUE?

Examples

```
x1 <- c(1, 3, 5, 9, 7)
any(is.na(x1))
```

```
[1] FALSE
```

```
all(x1 > 2)
```

```
[1] FALSE
```

```
which(x1 > 6)
```

```
[1] 4 5
```

4.4 Set Functions

Function	Description
<code>union(A, B)</code>	$A \cup B$
<code>intersect(A, B)</code>	$A \cap B$
<code>setdiff(A, B)</code>	$A \setminus B$
<code>setequal(A, B)</code>	Is $A = B$?
<code>is.element(x, A)</code>	$x \in A$?

4.5 Combinatorics

Function	Description
<code>factorial(x)</code>	Returns $x!$ (number of permutations).
<code>choose(n, k)</code>	Binomial coefficient, i.e. $\binom{n}{k}$.
<code>combn(x, m)</code>	Generates all combinations with m objects of the elements in x .
<code>sample(x, size, replace)</code>	Randomly selects elements.

4.6 Basic Descriptive Statistics

Functions

Function	Description
<code>mean(x)</code>	Arithmetic mean.
<code>median(x)</code>	Median.
<code>sd(x)</code>	Standard deviation.
<code>var(x)</code>	Variance.
<code>fivenum(x)</code>	Tukey's five number statistics.
<code>min(x)</code>	Minimum.
<code>max(x)</code>	Maximum.
<code>quantile(x, probs)</code>	Quantiles. Use <code>type=2</code> to get the same results as from <code>fivenum</code> for quartiles.
<code>cor(x)</code>	Correlation.
<code>cov(x)</code>	Covariance.

Examples

```
set.seed(789); x <- rnorm(15)
```

```
x[1:4]
```

```
[1]  0.52409671 -2.26076788 -0.01967972  0.18313989
```

```
fivenum(x)
```

```
[1] -2.26076788 -0.57710626 -0.36135148  0.08173009  0.92790739
```

```
quantile(x, probs=seq(0, 1, 0.2))
```

```
          0%          20%          40%          60%          80%          100%
-2.2607679 -0.7336064 -0.4351714 -0.1764192  0.2513313  0.9279074
```

4.7 Probability Distributions

All functions dealing with probability distributions always consist of a prefix (d, p, q or r) plus the (abbreviated) name of the distribution (e.g. norm, unif, chisq, etc.).

Example 1

We want to draw five numbers from a χ^2 distribution with 3 degrees of freedom:

```
set.seed(123); rchisq(n=5, df=3)
```

```
[1] 1.03611518 5.08870916 0.04818784 2.26693313 6.90085393
```

Example 2

If the height of male students follows a normal distribution with $\mu = 174$ cm and $\sigma = 7$ cm, what percentage is taller than 180 cm? (Solution: ≈ 19.57 %)

```
pnorm(180, mean=174, sd=7, lower.tail=FALSE)
```

```
[1] 0.195683
```

4.8 Notes on Missing Values

Missing Values

- The function `is.na(x)` returns TRUE for missing and FALSE for non-missing values in a vector **x**. Therefore you can easily count the missing values by using `is.na` and the function `sum`.
- Other comparison operators (`==`, `!=`) can **not** be used to detect missing values (Comparisons involving missing values always return NA).
- The function `length` counts **all** values in a vector (regardless of their “missing status”).
- Functions such as `mean` or `sd` fail (return NA) whenever there is at least one value missing - unless you specify `na.rm=TRUE` in the function call.
- The `table` command (which creates a frequency table) by default will **ignore** missing values unless you specify `useNA="always"` or `useNA="ifany"` in the function call.
- Sometimes missing values are coded as **999** (or similar) in your data. You can replace them using the `which` command.

Example

```
# weight... weight of students in kilos, 999 means "missing" here.
weight <- c(79, 88, 59, 999, 91, 60)
mean(weight)
```

```
[1] 229.3333
```

Replace **999** by NA

```
weight[which(weight==999)] <- NA
weight
```

```
[1] 79 88 59 NA 91 60
```

```
mean(weight)
```

```
[1] NA
```

```
mean(weight, na.rm=TRUE)
```

```
[1] 75.4
```

5 Groupwise Descriptive Statistics

5.1 Properties of the Function `doBy::summaryBy`

1. Install the package `doBy`
2. We will use the function `summaryBy` to create summary tables that contain exactly the information which we want to see.

Syntax of <code>doBy::summaryBy</code>
<code>doBy::summaryBy(var(s) ~ groupvar(s), data=..., FUN=...)</code>

Notes:

- If there are more variables in the “formula”, use `+` to separate them.
- It is possible to use built-in functions as well as your own functions.
- The function `length` applied to **any** column of your data counts the observations.
- If there are more functions to apply groupwise to your data.frame, pack the function names in a list, i.e. `FUN=list(fun1, fun2, ...)`.
- The function `doBy::summaryBy` returns a handy `data.frame`!

5.2 Examples

Creating Sample Data

```
set.seed(123)
group <- rep(c("A", "B"), 30)
treatment <- rep(c("group1", "group2", "group3"), 10)
values <- rchisq(30, df=5)
mydata <- data.frame(group, treatment, values)
print(head(mydata, 5))
```

	group	treatment	values
1	A	group1	2.5718020
2	B	group2	8.0747086
3	A	group3	0.6485141
4	B	group1	4.3740386
5	A	group2	10.3216603

Example 1

Average of *values* by *treatment*:

```
summary1 <- doBy::summaryBy(values ~ treatment,  
data=mydata, FUN=mean)
```

Registered S3 method overwritten by 'quantmod':

```
method          from  
as.zoo.data.frame zoo  
print(summary1)
```

```
      treatment values.mean  
1      group1      3.207749  
2      group2      5.004441  
3      group3      4.296330
```

Example 2

Minimum, maximum and a frequency count of *values* by *treatment* and *group*:

```
summary2 <- doBy::summaryBy(values ~ treatment + group,  
data=mydata, FUN=list(min, max, length))  
# change name of last column in summary table:  
names(summary2)[5] <- "N"  
print(summary2)
```

```
      treatment group values.min values.max  N  
1      group1      A  1.2220565   3.208945 10  
2      group1      B  1.3825887   6.184881 10  
3      group2      A  1.4285405  10.321660 10  
4      group2      B  0.6062728   8.500349 10  
5      group3      A  0.6485141   8.211414 10  
6      group3      B  3.1459618   5.409890 10
```

6 Graphs

6.1 Create Sample Data

```
x1 <- rep(c("A", "B"), 50)
set.seed(111)
x2 <- sample(c("U", "V", "W"), 100, replace=TRUE)
set.seed(345)
x3 <- rnorm(100, 100, 15)
set.seed(567)
x4 <- rchisq(100, 3)
X <- data.frame(x1, x2, x3, x4)
head(X, 3)
```

	x1	x2	x3	x4
1	A	V	88.22638	3.5291558
2	B	W	95.80728	2.7358626
3	A	W	97.57813	0.9237687

6.2 Colours

Here are some ways to specify colors in your plot:

1. Colour name (use `colours()` to display all available colors)
2. As an RGB or HEX value.
3. Using colour palettes from external libraries.

6.3 Saving Graphs to a File

```
pdf("filename.pdf") # or png(...), jpeg(...), ...
# graphics commands here
dev.off()
```

If you want to save output from a statistical procedure, you can use

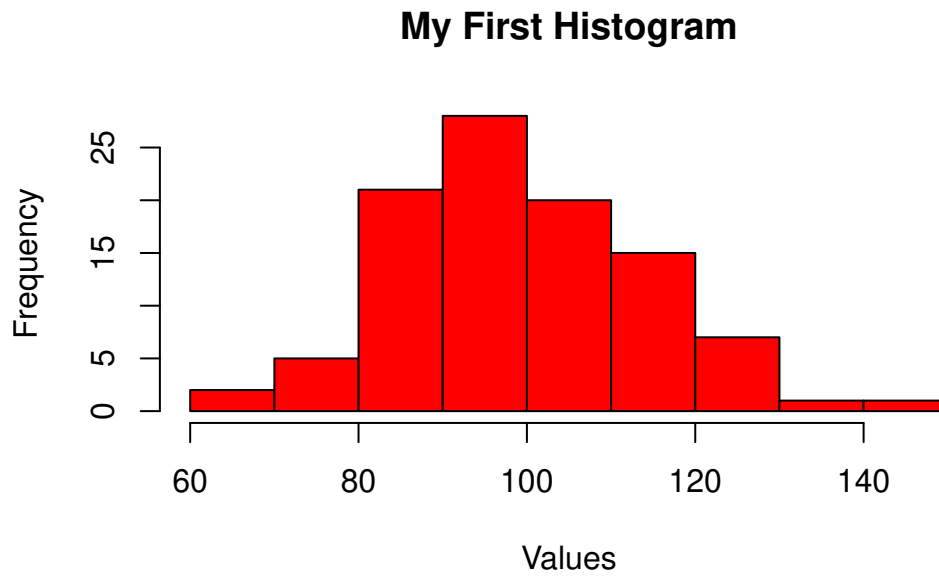
```
sink("filename.txt")
# ...statistics...
sink()
```

6.4 Example: Histogram

```
head(X, 3)
```

	x1	x2	x3	x4
1	A	V	88.22638	3.5291558
2	B	W	95.80728	2.7358626
3	A	W	97.57813	0.9237687

```
hist(X$x3, col=rgb(1,0,0), main="My First Histogram", xlab="Values")
```



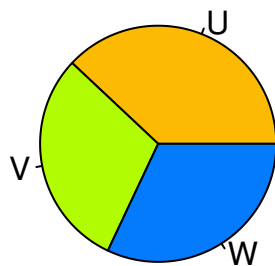
6.5 Example: Pie Chart

```
head(X, 3)
```

```
  x1 x2      x3      x4
1  A  V 88.22638 3.5291558
2  B  W 95.80728 2.7358626
3  A  W 97.57813 0.9237687
```

```
mytab <- table(X$x2) # Create frequency table first!
pie(mytab, col=c("#fcb303", "#b1fc03", "#037bfc"),
    main="A Simple Pie Chart")
```

A Simple Pie Chart

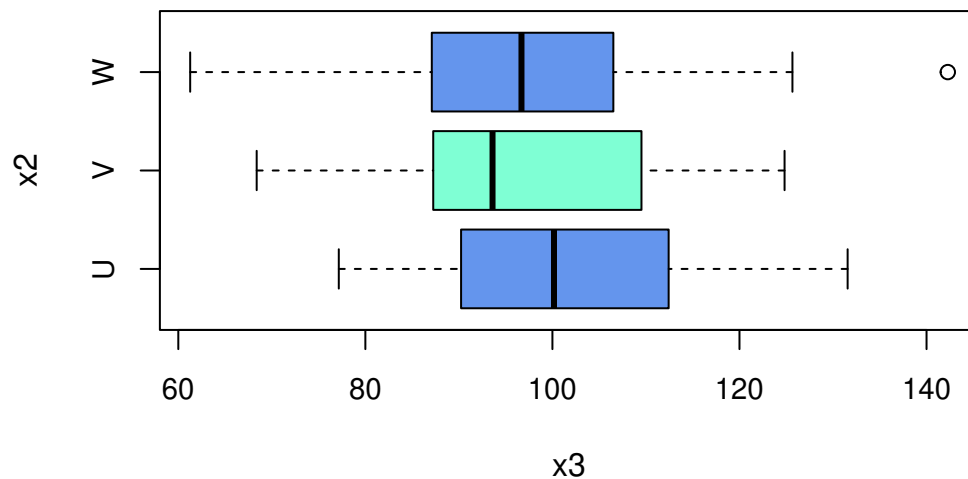


6.6 Example: Grouped Boxplot

```
head(X, 3)
```

```
  x1 x2      x3      x4
1  A  V 88.22638 3.5291558
2  B  W 95.80728 2.7358626
3  A  W 97.57813 0.9237687
```

```
boxplot(x3 ~ x2, data=X, horizontal=TRUE,
col = c("cornflowerblue", "aquamarine"), cex.axis=0.9)
```



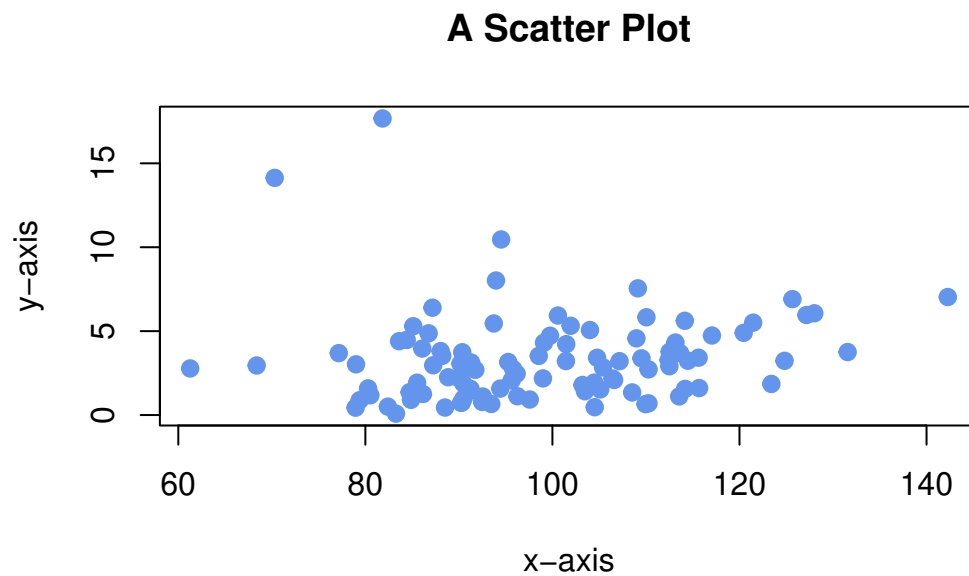
Note that we have three categories but we only passed a vector with two different colours. In this case, the vector is “recycled” here.

6.7 Example: Scatter Plot

```
head(X, 3)
```

```
  x1 x2      x3      x4
1  A  V 88.22638 3.5291558
2  B  W 95.80728 2.7358626
3  A  W 97.57813 0.9237687
```

```
plot(X$x3, X$x4, col="cornflowerblue", lwd=2, pch=19, xlab="x-axis",
     ylab="y-axis", main="A Scatter Plot")
```

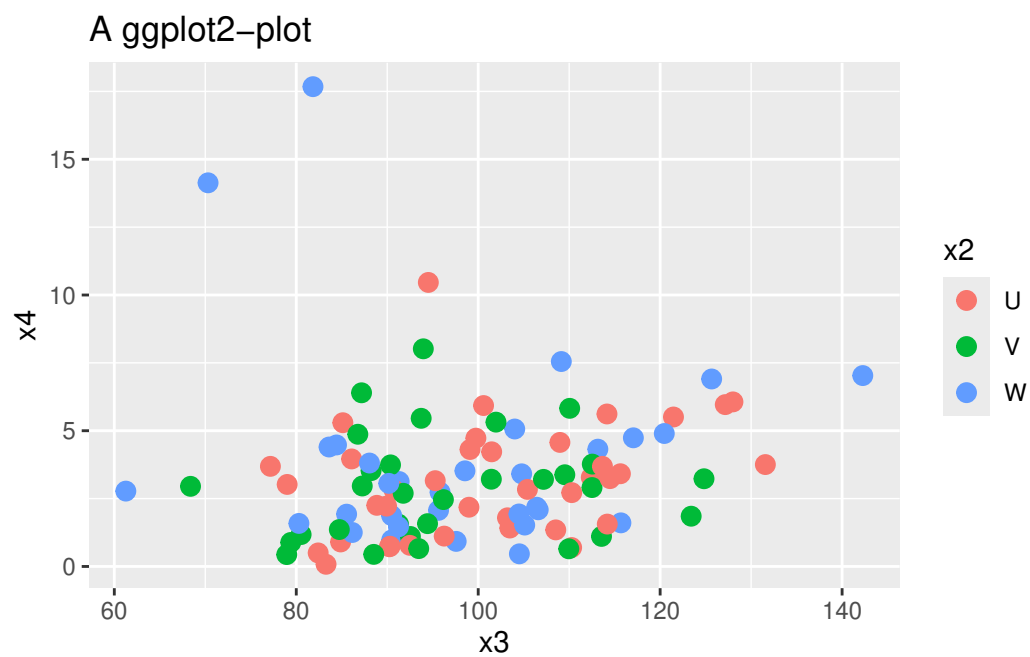


6.8 Example: Scatter Plot - ggplot2

```
head(X, 3)
```

```
  x1 x2      x3      x4
1  A  V 88.22638 3.5291558
2  B  W 95.80728 2.7358626
3  A  W 97.57813 0.9237687
```

```
library(ggplot2)
plt <- ggplot(X, aes(x=x3, y=x4, color=x2))
plt <- plt + geom_point(size=3)
plt <- plt + ggtitle("A ggplot2-plot")
plt
```



7 Summarizing Data Using summarytools

7.1 Create Sample Data

```
x1 <- rep(c("A", "B"), 50)
set.seed(111)
x2 <- sample(c("U", "V", "W"), 100, replace=TRUE)
set.seed(345)
x3 <- rnorm(100, 100, 15)
set.seed(567)
x4 <- rchisq(100, 3)
X <- data.frame(x1, x2, x3, x4)
head(X, 3)
```

	x1	x2	x3	x4
1	A	V	88.22638	3.5291558
2	B	W	95.80728	2.7358626
3	A	W	97.57813	0.9237687

7.2 Example: `summarytools::descr()`

```
summarytools::descr(X)
```

Non-numerical variable(s) ignored: x1, x2

Descriptive Statistics

X

N: 100

	x3	x4
-----	-----	-----
Mean	98.86	3.33
Std.Dev	14.73	2.67
Min	61.27	0.08
Q1	88.37	1.54
Median	96.22	2.96
Q3	109.74	4.36
Max	142.27	17.67
MAD	15.08	2.10
IQR	21.19	2.79
CV	0.15	0.80
Skewness	0.29	2.41
SE.Skewness	0.24	0.24
Kurtosis	-0.07	9.03
N.Valid	100.00	100.00
N	100.00	100.00
Pct.Valid	100.00	100.00

7.3 Example: summarytools::dfSummary()

```
summarytools::dfSummary(X, varnumbers=FALSE, valid.col=FALSE)
```

Data Frame Summary

X

Dimensions: 100 x 4

Duplicates: 0

Variable	Stats / Values	Freqs (% of Valid)	Graph	Missing
x1	1. A	50 (50.0%)	IIIIIIIIII	0
[character]	2. B	50 (50.0%)	IIIIIIIIII	(0.0%)
x2	1. U	38 (38.0%)	IIIIIII	0
[character]	2. V	30 (30.0%)	IIIIII	(0.0%)
	3. W	32 (32.0%)	IIIIII	
x3	Mean (sd) : 98.9 (14.7)	100 distinct values	:	0
[numeric]	min < med < max:		: : .	(0.0%)
	61.3 < 96.2 < 142.3		: : : .	
	IQR (CV) : 21.2 (0.1)		: : : :	
			. : : : : : :	
x4	Mean (sd) : 3.3 (2.7)	100 distinct values	. :	0
[numeric]	min < med < max:		: :	(0.0%)
	0.1 < 3 < 17.7		: : .	
	IQR (CV) : 2.8 (0.8)		: : :	
			: : : .	

8 Functions and Control Flow by Example

8.1 Loops with for

```
for (i in 1:3)
{
    print("Hello")
}
```

```
[1] "Hello"
```

```
[1] "Hello"
```

```
[1] "Hello"
```

8.2 Loops with repeat

```
n <- 1
repeat
{
    print(n)
    if (n >= 5)
    {
        break
    }
    n <- n + 1
}
```

```
[1] 1
```

```
[1] 2
```

```
[1] 3
```

```
[1] 4
```

```
[1] 5
```

8.3 Loops with repeat and break

```
n <- 1
repeat {

  if (n == 3) {
    n <- n + 1
    next
  }
  print(n)

  if (n >= 5) {
    break
  }

  n <- n + 1
}
```

```
[1] 1
[1] 2
[1] 4
[1] 5
```

8.4 Loops with while

```
i <- 1
while (i <= 3)
{
  print("Hello")
  i <- i + 1
}
```

```
[1] "Hello"
[1] "Hello"
[1] "Hello"
```

8.5 Conditional Execution - Two Branches

```
x <- 2
if (x == 1) {
  print("x is one")
} else {
  print("x is something else")
}
```

```
[1] "x is something else"
```

8.6 Conditional Execution - More Branches

```
x <- 2
if (x == 1) {
  print("x is one")
} else if (x == 2) {
  print("x is two")
} else if (x == 3) {
  print("x is three")
} else {
  print("hm...")
}
```

```
[1] "x is two"
```

8.7 Conditional Execution - Using switch

```
x <- 2

message <- switch(x,
                  "Value equals 1",
                  "Value equals 2",
                  "Value equals 3",
                  "Value is something else"
                  )
```

```
cat(message)
```

```
Value equals 2
```

8.8 Functions

```
squareandroundme <- function(x)
{
  result <- x * x
  result <- round(result, 1)
  return(result)
}
```

```
test <- squareandroundme(7.123)
```

```
test
```

```
[1] 50.7
```

9 Recommended Reading

- Kabacoff, R. (2011). *R in Action. Data Analysis and Graphics with R*. Shelter Island: Manning Publications.
- Ligges, U. (2008). *Programmieren mit R*. (3. Auflage). Berlin/Heidelberg: Springer.
- Sauer, S. (2019). *Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren*. Wiesbaden: Springer.
- Wickham, H., Golemund, G. (2016). *R for Data Science. Import, tidy, transform, visualize and model data*. Sebastopol: O'Reilly Media.